WIDS Project: SAT-Based Solvers for Constraint Problems

Majid husain

January 23, 2026

1 Introduction

This document serves as an extended explanation of my WIDS project submission. It is meant to elaborate on the repository README by describing the resources used, the approach followed while completing the project, the structure of the Git repository, and the key learnings gained from this work.

The main goal of this project is to explore the use of **SAT solvers** for solving constraint satisfaction problems such as Sudoku and Sokoban (A japanese block game). To build a strong conceptual foundation, I first implemented basic versions of the **DPLL** and **CDCL** algorithms and later used the ideas learned from these implementations while working on the problem-specific solvers.

2 Resources and Theory Used

The theoretical background required for this project was obtained primarily from course material and publicly available learning resources.

- **CS Logic Course Slides (Lectures 1–10):** These slides were used to understand propositional logic, CNF representations, SAT problems, and the basic DPLL algorithm.
- **Lectures by Prof. Anshuman Gupta:** These lectures helped in understanding the intuition behind DPLL, its optimizations, and the motivation for conflict-driven clause learning.
- **freeCodeCamp (Python):** Used to learn and revise Python at a basic level, focusing on clean syntax and program structure.

3 Project Development Approach

The project was completed in stages.

Initially, I implemented a **basic DPLL solver** to understand recursive backtracking, unit propagation, and satisfiability checking. After this, I implemented a **basic CDCL solver** to study conflict detection, clause learning, and non-chronological backtracking.

Once these foundational algorithms were implemented, I applied the same ideas to encode and solve constraint problems such as Sudoku and Sokoban using SAT-based approaches. This incremental approach helped ensure that the final solvers were grounded in a clear understanding of the underlying logic.

4 Repository Structure

The Git repository is organized into individual Python files, each corresponding to a specific part of the project.

- `q1.py`: SAT solver for Sudoku
- `q2.py`: SAT solver for Sokoban
- `dpll.py`: Basic implementation of the DPLL algorithm
- `cdcl.py`: Basic implementation of the CDCL algorithm

Each file is self-contained and can be accessed directly from the repository. The problem-specific solvers were written after implementing the standalone DPLL and CDCL algorithms, which helped in reusing concepts and maintaining clarity.

The commit history may appear inconsistent due to multiple iterative uploads and initial repository setup. For evaluation purposes, only the final versions of the files present in the repository should be considered.

5 How the System Works

The overall workflow of the project is as follows:

- Constraint problems such as Sudoku and Sokoban are first encoded into propositional logic formulas.
- These formulas are represented in Conjunctive Normal Form (CNF).
- A SAT-solving approach based on DPLL or CDCL is then used to determine whether the formula is satisfiable.
- If a satisfying assignment is found, it is interpreted back into a valid solution for the original problem.

This separation between problem encoding and SAT solving makes the approach modular and conceptually clean.

6 Learnings from the Project

This project provided hands-on experience with several important concepts:

- Understanding how SAT solvers work internally
- Translating real-world constraints into logical formulas
- Implementing recursive algorithms with backtracking
- Writing modular and readable Python code

Implementing DPLL and CDCL from scratch significantly improved my understanding of propositional logic and search-based algorithms.

SAT solvers are a nice tool for optipization and constraint satisfaction problems.

7 Conclusion

The project was fun :P